

# An Asynchronous Parallel Approach to Sparse Recovery

Deanna Needell

Department of Mathematical Sciences  
Claremont McKenna College  
Claremont, CA 91711, USA  
Email: dneedell@cmc.edu

Tina Woolf

Institute of Mathematical Sciences  
Claremont Graduate University  
Claremont, CA 91711, USA  
Email: tina.woolf@cgu.edu

**Abstract**—Asynchronous parallel computing and sparse recovery are two areas that have received recent interest. Asynchronous algorithms are often studied to solve optimization problems where the cost function takes the form  $\sum_{i=1}^M f_i(x)$ , with a common assumption that each  $f_i$  is *sparse*; that is, each  $f_i$  acts only on a small number of components of  $x \in \mathbb{R}^n$ . Sparse recovery problems, such as compressed sensing, can be formulated as optimization problems, however, the cost functions  $f_i$  are *dense* with respect to the components of  $x$ , and instead the signal  $x$  is assumed to be sparse, meaning that it has only  $s$  non-zeros where  $s \ll n$ . Here we address how one may use an asynchronous parallel architecture when the cost functions  $f_i$  are not sparse in  $x$ , but rather the signal  $x$  is sparse. We propose an asynchronous parallel approach to sparse recovery via a stochastic greedy algorithm, where multiple processors asynchronously update a vector in shared memory containing information on the estimated signal *support*. We include numerical simulations that illustrate the potential benefits of our proposed asynchronous method.

## I. INTRODUCTION

Technological advances in data gathering systems have led to the rapid growth of big data in diverse applications. At the same time, the recent emergence of inexpensive multicore processors, with the number of cores on each workstation on the rise, has motivated the study of parallel computing strategies. This has presented a challenge to many existing and popular algorithms that are designed to run iteratively and sequentially.

One possible approach to this problem is *synchronous* parallel computing, which assigns tasks to multiple cores and then waits for all cores to complete before the next step begins. Of course, the drawback of this approach is that all cores must wait for the slowest core to finish, even if the remaining cores all complete their computation quickly. An alternative approach, and one that has received much recent interest, is *asynchronous* parallel computing. In an asynchronous system, all cores run continuously, thus eliminating the idle time present in the synchronous approach, and all cores have access to shared memory and are able to make updates as needed.

Asynchronous algorithms are often studied to solve optimization problems where the cost function takes the form  $\sum_{i=1}^M f_i(x)$ . The decision variable  $x \in \mathbb{R}^n$  is updated iteratively, and its current state is accessible in shared memory by all processors. Many approaches to asynchronous parallel

computing for solving optimizations problems of this form assume that each  $f_i$  is sparse, which means that each  $f_i$  acts only on a small number of components of  $x$ ; therefore, this implies that individual core computations only depend on and update a small number of coordinates of  $x$  (e.g., [11], [25]). The benefit of this setup is that memory overwrites are rare, making it unlikely for the progress of faster cores to be overwritten by updates from slower cores.

There has also been a surge of interest in sparse recovery problems; for instance, literature in *compressed sensing* (e.g., [4], [5], [10]) addresses the problem of recovering a sparse vector  $x \in \mathbb{R}^n$  from few nonadaptive, linear, and possibly noisy measurements of the form  $y = Ax + z$ , where  $A \in \mathbb{R}^{m \times n}$  is the measurement matrix and  $z \in \mathbb{R}^m$  is noise. Recovering  $x$  from the noisy measurements  $y$  can be formulated as the optimization problem,

$$\min_{\tilde{x} \in \mathbb{R}^n} \frac{1}{2m} \|y - A\tilde{x}\|_2^2 \quad \text{subject to} \quad \|\tilde{x}\|_0 \leq s. \quad (1)$$

It is then natural to ask whether we can apply asynchronous parallel computing to solve (1). The challenge, however, is that the cost function depends on  $A$ , which is typically not taken to be sparse (e.g.,  $A$  is commonly taken to have standard i.i.d. Gaussian entries). This is in stark contrast to the typical assumptions made in the asynchronous parallel computing literature since the cost function is *dense* in the components of  $x$ , and instead the signal  $x$  is assumed to be sparse. Indeed, since the signal  $x$  is sparse, it is very likely that the same non-zero entries will be updated from one iteration to the next, while the remaining entries are set to zero to maintain a sparse solution. If executed asynchronously, then memory overwrites would be frequent, and a slow core could easily “undo” the progress of previous updates by faster cores.

**Contribution:** In this paper, we consider one of the stochastic greedy algorithms studied in [22] for sparse recovery. Focusing on the compressed sensing problem, we propose a strategy for utilizing the algorithm asynchronously despite the matrix  $A$  (and thus the cost function) not being sparse. Instead of having the current solution estimate in shared memory, a current estimate of the location of the non-zeros of the signal will be shared and available to each core. Thus, we provide a solution that merges asynchronous parallel architectures with

sparse recovery problems that have iterative updates which are not sparse, and will be a springboard to other more general approaches.

## II. RELATION TO PRIOR WORK

The seminal text [2] provides foundational work for parallel and distributed optimization algorithms. More recently, [25] studies an asynchronous variant of stochastic gradient descent called HOGWILD!. A key assumption in their analysis is that the cost function of the optimization problem is sparse with respect to the decision variable, meaning that most gradient updates only modify small and distinct parts of the solution. Much of the recent literature on asynchronous parallel computing borrows from the framework proposed in [25]. [11] also studies stochastic optimization when the cost function is sparse, and proposes two asynchronous algorithms with their analysis influenced by that in [25]. Also following the technique in [25], [19] presents an asynchronous parallel variant of the randomized Kaczmarz algorithm for solving the linear system  $Ax = y$  when  $A$  is large and sparse. [1] is interested in the same linear system with  $A$  symmetric positive definite and presents an asynchronous solver; note that the convergence rate analysis again depends on the sparsity of the matrix. Asynchronous parallel stochastic coordinate descent algorithms (which are clearly not designed for sparse solutions) are proposed in [18] and [17], which also follow the model of [25]. Other recent and relevant work on asynchronous parallel algorithms and analysis frameworks includes [6]–[9], [12]–[16], [20], [24], [27].

In the sparse recovery literature, popular greedy recovery algorithms include IHT [3], OMP [26], and CoSaMP [21]. Most relevant to our work is IHT, which we briefly review here. Starting with an initial estimation  $x^1 = 0$ , the IHT algorithm computes the following recursive update,

$$x^{t+1} = \mathcal{H}_s(x^t + A^*(y - Ax^t)), \quad (2)$$

where  $\mathcal{H}_s(a)$  is the thresholding operator that sets all but the largest (in magnitude)  $s$  coefficients of  $a$  to zero. The work [22] proposes a stochastic variant of IHT called StoIHT, which is the algorithm of focus here. Note that [22] also proposes a stochastic variant of GradMP [23] which is based on CoSaMP; our work here can easily be generalized to these other algorithms as well.

## III. ASYNCHRONOUS SPARSE RECOVERY WITH TALLY UPDATES

**Notation:** We denote by  $[M]$  the set  $\{1, 2, \dots, M\}$  and let  $p(1), \dots, p(M)$  be the probability distribution of an index  $i$  selected at random from the set  $[M]$ , so that  $\sum_{i=1}^M p(i) = 1$ . Let  $A^*$  denote the conjugate transpose of the matrix  $A$ . For a vector  $a \in \mathbb{R}^n$ , let  $\text{supp}_s(a)$  be the operator that returns the set of cardinality  $s$  containing the indices of the largest (in magnitude) elements of  $a$ . For a set  $\Gamma$ , let  $a_\Gamma$  denote the vector  $a$  with everything except the components indexed in  $\Gamma \subset \{1, \dots, n\}$  set to zero.

As described above, suppose we observe  $y = Ax + z$ , where  $x$  is  $s$ -sparse and supported on  $T \subset \{1, \dots, n\}$  (that is,  $\|x\|_0 = |\{i : x_i \neq 0\}| = |T| \leq s \ll n$ ). To recover  $x$  from the noisy measurements  $y$ , we aim to solve the optimization problem (1). Note that we can also express the cost function in (1) as

$$\frac{1}{2m} \|y - A\tilde{x}\|_2^2 = \frac{1}{M} \sum_{i=1}^M \frac{1}{2b} \|y_{b_i} - A_{b_i} \tilde{x}\|_2^2,$$

where  $y$  has been decomposed into non-overlapping vectors  $y_{b_i}$  of size  $b$ ,  $A_{b_i}$  has been decomposed into non-overlapping  $b \times n$  submatrices of  $A$ , and  $M = m/b$  (which for simplicity we assume is integral). Notice that the cost function now takes the form  $\sum_{i=1}^M f_i(x)$ , and each  $f_i(x) = \frac{1}{2bM} \|y_{b_i} - A_{b_i} x\|_2^2$  accounts for a block of observations of size  $b$ . The StoIHT algorithm from [22] for solving (1), specialized to the compressed sensing setting, is shown in Algorithm 1, where  $\gamma$  denotes a step-size parameter. The recovery error of the algorithm depends on the block size  $b$ ; we refer the reader to [22] for details.

---

### Algorithm 1 StoIHT Algorithm [22]

---

**input:**  $s, \gamma, p(i)$ , and stopping criterion  
**initialize:**  $x^1$  and  $t = 1$   
**repeat**  
    **randomize:** select  $i_t \in [M]$  with probability  $p(i_t)$   
    **proxy:**  $b^t = x^t + \frac{\gamma}{Mp(i_t)} A_{b_{i_t}}^* (y_{b_{i_t}} - A_{b_{i_t}} x^t)$   
    **identify:**  $\Gamma^t = \text{supp}_s(b^t)$   
    **estimate:**  $x^{t+1} = b_{\Gamma^t}^t$   
                   $t = t + 1$   
**until** halting criterion *true*  
**output:**  $\hat{x} = x^t$

---

In the asynchronous approach, each core will execute its own slightly modified version of Algorithm 1. In order to avoid having each core update the entire solution in shared memory at each iteration, we propose to instead keep a *tally* vector  $\phi \in \mathbb{R}^n$  in shared memory. The tally  $\phi$  will contain information on the estimated support locations identified by each core's most recent iteration.

The steps executed by *each* core at each iteration are detailed in Algorithm 2, where the tally vector  $\phi$  is available for read and write by each core. Note that in Algorithm 2 the iteration number  $t$  and the iterate  $x^t$  are local to each core. Also note that in the tally update step, the tally is incremented by  $t$  (the core's local iteration number) on  $\Gamma^t$  and decremented by  $t - 1$  on  $\Gamma^{t-1}$  (as mentioned in [25], these operations can be performed atomically on most modern processors). This is to provide more weight to faster cores that are further along in the algorithm and should thus be able to provide a better support estimate, and less weight to slower cores. In order to maintain only the information from each core's most recent iteration, the tally contribution from the previous iteration  $t - 1$  is removed.

It is worth mentioning that *inconsistent* reads, where components of the shared memory vector variables may be written by some cores while being simultaneously read by others, is discussed in the current literature on asynchronous computing. It is desirable to incorporate this feature into any models and analyses to more faithfully represent modern computational architectures. Our approach is not immune to inconsistent reads of the tally  $\phi$  by any means, however, the hope is that the algorithm will be more robust to inconsistent reads of  $\phi$  than a current solution estimate since its use in the algorithm is more passive, and that an inconsistently read version of  $\phi$  will still provide valuable information on the support locations of the signal. Although not addressed here, analyses and simulations of this impact are certainly of interest.

---

**Algorithm 2** Asynchronous StoIHT Iteration

---

Each core performs the following at each iteration. The tally vector  $\phi$  is available to each core.

**randomize:** select  $i_t \in [M]$  with probability  $p(i_t)$   
**proxy:**  $b^t = x^t + \frac{\gamma}{Mp(i_t)} A_{b_{i_t}}^* (y_{b_{i_t}} - A_{b_{i_t}} x^t)$   
**identify:**  $\Gamma^t = \text{supp}_s(b^t)$   
 $\tilde{T}^t = \text{supp}_s(\phi)$   
**estimate:**  $x^{t+1} = b^t_{\Gamma^t \cup \tilde{T}^t}$   
**update tally:**  $\phi_{\Gamma^t} = \phi_{\Gamma^t} + t$   
 $\phi_{\Gamma^{t-1}} = \phi_{\Gamma^{t-1}} - (t-1)$   
 $t = t + 1$

---

#### IV. SIMULATIONS

In this section, we provide encouraging experimental results for the proposed asynchronous tally update scheme. In all of the experiments, we take the signal dimension  $n = 1000$ , the sparsity level  $s = 20$ , the number of measurements  $m = 300$ , the block size  $b = 15$ , the step-size  $\gamma = 1$ , and the initial estimate  $x^1 = 0$ . The algorithms exit once  $\|y - Ax^t\|_2$  drops below the tolerance  $10^{-7}$  or a maximum of 1500 iterations is reached.

##### A. StoIHT with an Accurate Support Estimate

Our first experiment provides evidence that performing the estimation step onto the top  $s$  coefficients in addition to a set that accurately describes the true signal support will increase the convergence rate of the standard StoIHT algorithm. That is, we execute Algorithm 1 with the modification that  $x^{t+1} = b^t_{\Gamma^t \cup \tilde{T}}$ , where  $\tilde{T}$  estimates the true support  $T$  with  $|\tilde{T}| = s$  and accuracy  $\frac{|\tilde{T} \cap T|}{|\tilde{T}|} = \alpha$ . Figure 1 compares the mean recovery error as a function of iteration over 50 trials of the standard StoIHT algorithm (Algorithm 1) with the modified StoIHT algorithm as described for various values of  $\alpha$ . Indeed, for  $\alpha > 0.5$ , fewer iterations are needed on average for the algorithm to converge. When  $\alpha = 1$ , on average roughly half as many iterations as the standard algorithm are needed for convergence. We emphasize that this experiment, as a proof of concept, has no parallel implementation, and hence iterations are comparable to runtime. This result suggests that

the asynchronous approach using Algorithm 2 could lead to speedups as long as the tally  $\phi$  becomes accurate fast enough.

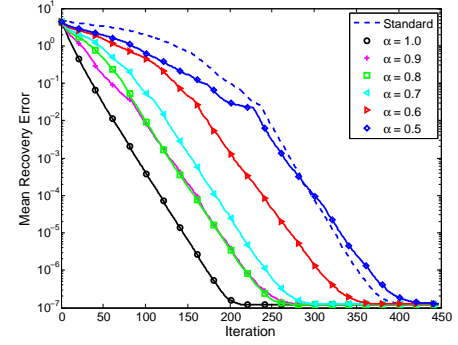


Fig. 1: Mean recovery error over 50 trials versus iteration of StoIHT and a modified version of StoIHT. In the modified StoIHT, we execute Algorithm 1 where the estimation step is performed by projecting  $b^t$  onto  $\Gamma^t \cup \tilde{T}$  at each iteration, where  $\tilde{T}$  has accuracy  $\frac{|\tilde{T} \cap T|}{|\tilde{T}|} = \alpha$ .

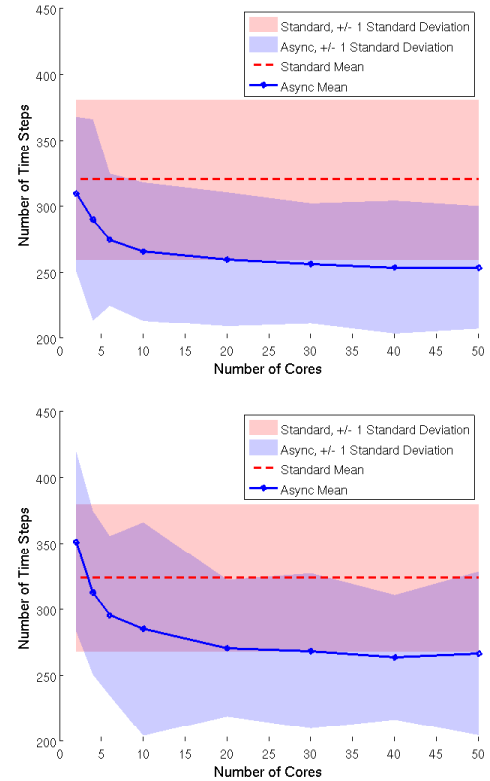


Fig. 2: Comparison of the number of time steps executed until convergence versus the number of cores used in the asynchronous StoIHT method. The heavy line denotes the mean number of time steps over 500 trials, and the boundaries of the shaded region indicate  $\pm 1$  standard deviation from the mean. (Upper) All cores are simulated to complete an iteration in a single time step; (lower) half of the cores are “slow” and complete an iteration only once out of every four time steps.

## B. Asynchronous StoIHT

Here, we simulate the execution of asynchronous StoIHT with  $c$  cores, where each core uses the iteration defined in Algorithm 2. For clarity, define a *time step* to be the amount of time needed for the fastest core to complete an iteration of Algorithm 2. First, we assume each core takes the same amount of time to perform an iteration; thus, in a single time step, all  $c$  cores complete an iteration of Algorithm 2. We also assume that an iteration of Algorithm 1 takes a single time step. When executing the  $t$ -th time step, and hence the  $t$ -th iteration for each core, every core utilizes the same set  $\hat{T}^t$  identified by the tally  $\phi$ . Once each core completes its estimation step, the tally is updated via  $\Gamma^t$  from *each* core. As soon as *any* core achieves the exit criteria at its local iteration  $t$ , the algorithm terminates, and  $t$  time steps are recorded as the number of time steps until exit is achieved. The upper plot of Figure 2 displays the mean number of time steps until exit, over 500 trials, for both the standard and asynchronous StoIHT algorithms, plotted against the number of cores used in the asynchronous method. The shaded regions indicate  $\pm 1$  standard deviation from the mean. Since the standard method does not depend on the number of cores, horizontal lines are shown. Notice that the mean number of time steps required in the asynchronous method is always less than the standard algorithm; therefore, a speedup in the total time for the algorithm to converge is expected when executed asynchronously.

Next, we modify the previous experiment to simulate the impact of slow cores. We take half of the cores to be “fast,” meaning that they continue to update the shared tally at each time step; the other half of the cores, however, are “slow” and only complete an iteration and update the tally once out of every four time steps. The lower plot of Figure 2 displays the mean number of time steps until exit versus the number of cores in the asynchronous method. For  $c = 2$ , no improvement is gained from the asynchronous method on average, however, improvement is observed for the larger values of  $c$  tested.

## V. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed an asynchronous parallel variant of an existing stochastic greedy algorithm for sparse recovery. Our method is distinct from much of the existing literature on asynchronous algorithms because: 1) sparsity assumptions on the cost function, which are common to the existing literature, are not necessary, and 2) the current solution iterate is not available in shared memory, but instead a *tally* vector containing the latest information from the cores on the estimated support of the signal is shared and utilized; this approach provides necessary robustness to asynchronous updates and inconsistent reads even when traditional updates cannot be made sparse.

A similar approach could also be applied to the second stochastic greedy algorithm studied in [22], namely, StoGradMP. Although we specialized to the compressed sensing problem, both StoIHT and StoGradMP are studied for general sparse recovery problems in [22]; thus, it would be

interesting to explore the proposed idea in other settings. It would also be beneficial to develop theory for the proposed approach, perhaps building from the theory in [22] and the current literature analyzing asynchronous algorithms, and include the incorporation of architecture realities such as inconsistent reads.

## ACKNOWLEDGMENT

The work of Deanna Needell was partially supported by NSF CAREER grant #1348721 and the Alfred P. Sloan Foundation. The work of Tina Woolf was partially supported by NSF CAREER grant #1348721.

## REFERENCES

- [1] H. Avron, A. Drusinsky, and A. Gupta. Revisiting asynchronous linear solvers: Provable convergence rate through randomization. *J. ACM*, 62(6):51, 2015.
- [2] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and distributed computation: numerical methods*, volume 23. Prentice Hall, Englewood Cliffs, NJ, 1989.
- [3] T. Blumensath and M. Davies. Iterative hard thresholding for compressive sensing. *Appl. Comput. Harmon. Anal.*, 27(3):265–274, 2009.
- [4] E. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inform. Theory*, 52(2):489–509, 2006.
- [5] E. Candès, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Comm. Pure Appl. Math.*, 59(8):1207–1223, 2006.
- [6] L. Cannelli, F. Facchinei, V. Kungurtsev, and G. Scutari. Asynchronous parallel algorithms for nonconvex big-data optimization: Model and convergence. *arXiv preprint arXiv:1607.04818*, 2016.
- [7] D. Davis. The asynchronous palm algorithm for nonsmooth nonconvex problems. *arXiv preprint arXiv:1604.00526*, 2016.
- [8] D. Davis, B. Edmunds, and M. Udell. The sound of apalm clapping: Faster nonsmooth nonconvex optimization with stochastic asynchronous palm. *arXiv preprint arXiv:1606.02338*, 2016.
- [9] A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Proc. Adv. in Neural Processing Systems (NIPS)*, pages 1646–1654, 2014.
- [10] D. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52(4):1289–1306, 2006.
- [11] J. Duchi, M.I. Jordan, and B. McMahan. Estimation, optimization, and parallelism when data is sparse. *Proc. Adv. in Neural Processing Systems (NIPS)*, pages 2832–2840, 2013.
- [12] J.C. Duchi, S. Chaturapruek, and C. Ré. Asynchronous stochastic convex optimization. *arXiv preprint arXiv:1508.00882*, 2015.
- [13] H.R. Feyzmahdavian, A. Aytekin, and M. Johansson. An asynchronous mini-batch algorithm for regularized stochastic optimization. *Proc. IEEE Conf. Decision and Control (CDC)*, pages 1384–1389, 2015.
- [14] Z. Huo and H. Huang. Asynchronous stochastic gradient descent with variance reduction for non-convex optimization. *arXiv preprint arXiv:1604.03584*, 2016.
- [15] R. Leblond, F. Pedregosa, and S. Lacoste-Julien. Asaga: Asynchronous parallel saga. *arXiv preprint arXiv:1606.04809*, 2016.
- [16] X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. *Proc. Adv. in Neural Processing Systems (NIPS)*, pages 2737–2745, 2015.
- [17] J. Liu and S.J. Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM J. Optimization*, 25(1):351–376, 2015.
- [18] J. Liu, S.J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *J. Machine Learning Research*, 16(285-322):1–5, 2015.
- [19] J. Liu, S.J. Wright, and S. Sridhar. An asynchronous parallel randomized Kaczmarz algorithm. *arXiv preprint arXiv:1401.4780*, 2014.
- [20] H. Mania, X. Pan, D. Papailiopoulos, B. Recht, K. Ramchandran, and M.I. Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *arXiv preprint arXiv:1507.06970*, 2015.

- [21] D. Needell and J. Tropp. CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *Appl. Comput. Harmon. Anal.*, 26(3):301–321, 2009.
- [22] N. Nguyen, D. Needell, and T. Woolf. Linear convergence of stochastic iterative greedy algorithms with sparse constraints. *arXiv preprint arXiv:1407.0088*, 2014.
- [23] N. H. Nguyen, S. Chin, and T. D. Tran. A unified iterative greedy algorithm for sparsity-constrained optimization. 2013. Submitted.
- [24] Z. Peng, Y. Xu, M. Yan, and W. Yin. Arock: an algorithmic framework for asynchronous parallel coordinate updates. *arXiv preprint arXiv:1506.02396*, 2015.
- [25] B. Recht, C. Ré, S. Wright, and F. Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Proc. Adv. in Neural Processing Systems (NIPS)*, pages 693–701, 2011.
- [26] J. Tropp and A. Gilbert. Signal recovery from partial information via orthogonal matching pursuit. *IEEE Trans. Inform. Theory*, 53(12):4655–4666, 2007.
- [27] S.Y. Zhao and W.J. Li. Fast asynchronous parallel stochastic gradient decent. *arXiv preprint arXiv:1508.05711*, 2015.